Euso Simulation and Analysis Framework

Coding Rules

Sep 15, 2004 - Version 1.0
Doc. Ref. EUSO-SDA-REP-XXX-Y

A. Thea [1]

Istituto Nazionale di Fisica Nucleare & Università di Genova
Italy

Abstract

The aim of this guide is to give an overview of the *ESAF* classes features and their integration in *ESAF* and ROOT. The conventions and the rules to include a new class in the framework taking advantage of the *ESAF* facilities will be described.

[1] e-mail: Alessandro.Thea@ge.infn.it

Contents

1 Introduction

This guide is miles away from to be complete and exhaustive. But *ESAF* developers team is getting bigger and bigger and I hope that an even small reference guide dedicated to the developers may be useful to avoid painful debugging and long and sometimes "hot" mail exchanges.

This first version of this guide would like to cover several aspects related with the classes *ESAF* is founded on.

2 Text formatting

Indentation: 4 spaces, no tabs.

2.1 Configuring VIM

Add to your `/.vimrc` the following lines:

```
set shiftwidth=4
set softtabstop=4
set tabstop=4
set expandtab
```

to tell VIm to use the correct indentation.

2.2 Configuring Emacs

Add to your `/.emacsrc` the following line:

```
(setq c-basic-offset 4)
```

3 Coding Conventions

ESAF follows ROOT/Taligent name convention. Due to historical reasons there some exceptions exists. Class names don't begin with a T and a restricted number of them begins with E.

The following paragraph is part of Addison-Wesley's *Taligent Guide* [4].

Select C++ identifiers (including types, functions, and classes) carefully. When a programmer sees a name, it might be out of context; choose names to enhance readability and comprehension. A name that seems cute or easy to type can cause trouble to someone trying to decipher code. Remember, code is read many more times than it is written; err on the side of long, readable names. Internal code names should not appear anywhere in the interfaces to the system. Even inside your implementation, it's better to use the prosaic form if there is one.

To make the scope of names explicit, Taligent uses the conventions of table 1.

In any name that contains more than one word, the first word follows the convention for the type of the name, and subsequent words follow with the first letter of each word capitalized, such as TTextBase. Do not use underscores except for `#define` symbols.

4 EsafConfigClass and configuration files

Many *ESAF* Classes are initialized via config files contained in `config/` directory. The interface between `.cfg` files and classes is handled by the `Config` singleton and `EusoConfigurable` objects.

All the classes inheriting from `EusoConfigurable` can access the parameters stored in the `.cfg` files using `EusoConfigurable::Conf()` interface. `Conf()` returns a pointer to the correct `EusoConfigFileParser` object stored in `Config`. Each class must be properly set to get the correct parser from `Conf()`.

This is done adding the following line in the class definition:

```
EsafConfigClass(Optics, ParamOpticalSystem)
```

`EsafConfigClass()` macro has two arguments, the first is `ClassType` and the second `ClassName`. `ClassType` refers to the `config/` subdirectory where the class config file is, while `ClassName` is the file name. In *ESAF* each class has its own config file.

A detailed description of *ESAF* configuration system can be found in [1]

5 ROOT integration

ESAF is deeply connected with ROOT. ROOT classes are used to provide the user with a user friendly Graphical User Interface (GUI), an efficient I/O system and the

Identifier	Convention	Example
Types	Begin with a capital letter	Boolean
Enumeration types	Begin with E	EFreezeLevel
Members	Begin with f for field1; functions begin with a capital letter	fViewList, DrawSelf()
Static variables	Begin with g; applies to static variables in functions and global variables (excluding static data members of a class)	gDeviceList
Static data members	Begin with fg; includes class globals	TView::fgTokenClient
Locals and parameters	Begin with a word whose initial letter is lowercase; local automatic variables only, treat statics like globals	seed, port, theCurrentArea
Constants	Begin with k; including names of enumeration constants and constant statics	kMenuCommand
Acronyms	All uppercase	TNBPName, not TNbpName
Template arguments	Begin with A	AType
Getters and setters	Begin with Set..., Get..., or Is... (Boolean); use sparingly	SetLast(), GetNext(), IsDone()
Allocator and adopters	Begin with Create..., Copy..., Adopt..., or Orphan...;	CreateName()

Table 1: *Taligent* conventions used in *ESAF*

automatic documentation generation. Integration of new classes in ROOT is done with `ClassDef`, `ClassImp` macros and `rootcint` dictionary generator. The `ClassDef` and `ClassImp` macros are necessary to link classes to the dictionary generated by CINT.

`rootcint` process class headers and parsing the class structure, builds the streamers needed for I/O. `LinkDef.hh` file in `include` directories tells rootcint for which classes to generate the method interface stubs. .

In the class header files the following line must be added to class header files:

```
ClassDef(ClassName,ClassVersionID)
```

The `ClassVersionID` is used by the ROOT I/O system. This feature is basically reserved to the classes in `packages/common/root/` which are used to store data in the rootfile, like `EPhoton`, must have `ClassVersionID >= 1`. Elsewhere standard ESAF classes do not need to be written on file therefore `ClassVersionID` must be equal to

0.

NOTE: The `ClassDef` macro must be the last item before the closing `};` in a class definition. It contains its own private and public tags so it can be added to either a private or public part of a class definition.

Similarly, in the source file must be added:

```
ClassImp(ClassName)
```

which provides the interface needed to generate the documentation.

`ClassDef`, `ClassImp`, ROOT I/O and RTTI systems are explained extensively in the ROOT User's Guide [3].

6 Logging and Message handling in *ESAF*

6.1 Messenger

TO_DO

6.2 TObject and ROOT message system

7 Classes coding conventions

7.1 Files

All files belonging to the ESAF CVS are required to have basic information stored in a standard header like the following:

```
// $Id: ClassFormat.tex,v 1.3 2004/09/29 10:07:43 thea Exp $
// Author: A.Thea   2004/07/19

/**************************************************************************
 * ESAF: Euso Simulation and Analysis Framework                           *
 *                                                                        *
 *  Id: ParamOpticalSystem                                                *
 *  Package: Optics                                                       *
 *  Coordinator: Alessandro.Thea                                          *
 *                                                                        *
 **************************************************************************/
```

`$Id: ClassFormat.tex,v 1.3 2004/09/29 10:07:43 thea Exp $`: CVS file informations.

`Author`: the name of the creator and the creation date. It is read by the automatic documentation system.

`Id`: string that identifies the file (i.e. the class name or the namespace name).

`Package`: the name of the package it belongs to.

`Coordinator`: the package coordinator identifier, $< FirstName > . < LastName >$.

The ESAF header must be present in header, source and config files (`.hh`, `.cc`, `.cfg`).

8 Comments

In a projects as big as *ESAF*, commenting the code is essential. Moreover, comments written in the right format can be intercepted and included them in the documentation. In *ESAF* we use this approach so each developer has to keep up-to-date only the code documentation and the automatic documentation system takes care of producing the package documentation.

For each class three different type of comments are required:

1. A header inside the header file,
2. Class and class' parameter description in the source file,
3. Class' member functions explanation in the source file.

8.1 Header files

A typical *ESAF* class looks like:

```
////////////////////////////////////////////////////////////////////////////
//                                                                        //
// ParamOpticalSystem                                                     //
//                                                                        //
// Parameterized optical system                                           //
//                                                                        //
////////////////////////////////////////////////////////////////////////////

class ParamOpticalSystem : public OpticalSystem {
public:
    ParamOpticalSystem();
```

(...continued)

```
    virtual ~ParamOpticalSystem();

private:

    enum EPsfType { kPointLike, kGaussian };

    void Init();
    Double_t *fIntegral;         // fNbins+1 size array containing the
                                 // line integral of the fs profile
    Int_t fNbins;                // size of fIntegral

    EsafConfigClass(Optics,ParamOpticalSystem)

    ClassDef(ParamOpticalSystem,1)
};
```

For each class or namespace, two comments are required: class/namespace title before its declaration, and class data members description.

8.1.1 Class title

Just before the class declaration there is a brief description of the class (few words). In case of a single class header may be redundant but becomes useful when several classes are defined in the same file.

```
//////////////////////////////////////////////////////////////////////////
//                                                                      //
// ParamOpticalSystem                                                   //
//                                                                      //
// Parameterized optical system                                         //
//                                                                      //
//////////////////////////////////////////////////////////////////////////
```

8.1.2 Data members

The descriptions must follow data member declaration, inline or in the following line.

Sometime special keywords can be found after `//`. These information are used by the dictionary generator `rootcint` Special comments (`ClassVersionID != 0`)like `EShower`, `EAtmosphere` and `EDetector`.

```
Double_t *fIntegral;
// fNbins+1 size array containing line integral of the fs profile

Int_t fNbins;                // size of fIntegral
```

8.2 Source files

8.2.1 Class Documentation

The information stored in the class documentation block is twofold. First, features and proposes are explained in detail. Then, if class uses *ESAF* configuration system, the list of the config file parameters follows.

```
//_____________________________________________________________________________
//   Parameterized Optical System
//   ============================
//
//   Parameterized optical system. This Optical System is done to reproduce the
//   behaviour of an optics design through a set of parameters.
//   The main pourpose is to simulate and OS with a given Triggering efficacy
//   without the need of the full MC simulation.
//
//   ParamOpticalSystem behaves as an ideal optical system with a given focal
//   surface.
//
//   The position an incoming photon hits the focal surface is calculated
//   accordingly the relation
//
//   d = (Dmax/ThetaMax)*theta
//
//   where:
//   d is the distance of the impact point on the FS from the center of the FS
//   calculated along the FS.
//
//   Dmax is the maximum distancs.
//
//   theta is the angle between the incoming photon direction and the optical
//   axis.
//
//   ThetaMax is the maximum value of theta accepted by the optics.
//
//   Config file parameters:
//   =======================
//
//   fPos.Z [mm] : Z coordinate of the bottom base of the OpticalSystem in
```

(...continued)

```
//   Detector reference System.
//
//   fPsfType [option]: type of PSF to use.
//   - options
//        point: No spread.
//        gauss: Gaussian psf. RMS and angles of incidence listed in
//               fGaussSpread.filename
//
```

Note that EVERYTHING until the first non-commented line is considered as a valid class description block.

Parameters can be numbers or strings. Unit of measure must be specified when the parameter is not pure numbers, otherwise the `[string]` keyword, in case of strings.

In the latter case all the valid options must be listed after the parameter description.

8.2.2 Member fuctions

A member function description block starts immediately after '`{`' and looks like this:

```
//_____________________________________________________________________________
Double_t ParamOpticalSystem::FocalSurfProfile( Double_t r ) const {
    //
    // Profile of the focal surface Z(r)
    //

    if ( r < 0 || r > fRmax ) return 0;

    return fFocalSurfShape->Eval(r);
}
```

Like in a class description block, EVERYTHING until the first non-commented line is considered as a valid member function description block.

A Acknowledgements

B References

References

[1] D. De Marco and M. Pallavicini, *Euso Simulation and Analysis Framework*; EUSO-SIM-ESAF-001-01, available on LiveLink

[2] The ROOT System http://root.cern.ch

[3] The ROOT User's Guide http://root.cern.ch/root/doc/RootDoc.html

[4] Taligent's Guide to Designing Programs Online version of the book, *Taligent's Guide to Designing Programs: Well-Mannered Object-Oriented Design in C++*; Addison-Wesley.

C CreateNewClass script

`CreateNewClass.pl` is a perl script to create empty class files based on ESAF class structure.

```
Usage: CreateNewClass.pl -n name [-s] [-u username] [-p parent]

                -n   The name of the class.
                -s   Create a singleton.
                -u   Include the author name in the class header.
                -p   Specify class parent if exists.
```

Each library is contained in a directory with two subdirectories, `include` and `src` To add a new class to a library, go to the library directory and execute `CreateNewClass.pl`.

`ClassName.hh` and `ClassName.cc` will be created in `include/` and `src/` subdirectories of the current directory. `esaf/packages/tools`

```
[ale]: CreateNewClass.pl
```

D CreateNewLib script

TO_DO